UNIDAD 9
PROGRAMACIÓN ORIENTADA A OBJETOS II
MARATÓN DE EJERCICIOS

PROGRAMACIÓN
CFGS DAM

Autores:
CEEDCV. WirtzJava
Modificado por:
Jesús A. Martínez

ver 0.1

Licencia

9.1. Realiza las siguientes operaciones:

- Crea una clase llamada Figura2D con atributos numéricos con decimales y públicos: ancho y alto y un método verDim que muestre por consola el alto y el ancho en una sola línea, con dos decimales.
- Define una clase llamada Triángulo que es hija de Figura2D y añádele una cadena llamada estilo (vale: isósceles, rectángulo, equilátero, etc.), un método llamado área que devuelva la superficie del triángulo y un último método llamado verEstilo que muestre por consola el valor del atributo estilo.
- Finalmente hacer un programa que cree un triángulo, asigne valores a sus atributos y use los métodos para ver sus dimensiones, estilo y área.

9.2. Copia las clases anteriores como una nueva versión de las mismas (añade sufijo _v2) y realiza los siguientes cambios:

- Ahora los atributos ancho y alto serán privados.
- Añade los métodos getter y setter para ambos atributos.
- Modifica la clase Triangulo obligados por los cambios en su clase padre.
- Haz una copia del programa anterior, sobre las nuevas clases creadas, y comprueba que el programa creado en el ejercicio anterior sigue funcionando.

Nota: Cuando copies las clases y les cambies el nombre, seguramente tendrás que cambiar y actualizar la clase padre de Triangulo_v2. Esto pasará siempre que copies y pegues clases.

9.3. Copia las clases anteriores como una nueva versión de las mismas (añade sufijo _v3) y realiza los siguientes cambios:

- Añade un constructor a la clase Triangulo al que se le pasan tres parámetros: estilo, alto y ancho.
- Se invocará al constructor por defecto de la clase base.
- Modificar el programa de los ejercicios anteriores para que los triángulos sean creados mediante este nuevo constructor.

9.4. Copia las clases anteriores como una nueva versión de las mismas (añade sufijo _v4) y realiza los siguientes cambios:

- Añade un constructor a la clase Figura2D al que se le pasan dos parámetros: alto y ancho. ¿Funciona ahora el constructor de Triangulo (creado en el ejercicio anterior)? ¿Por qué?

- Reescribe el constructor de Triangulo creado en el ejercicio anterior para que haga uso del nuevo constructor de la clase base.
- Comprueba que el programa creado en el ejercicio anterior sigue funcionando.
- Haz una copia del programa anterior, sobre las nuevas clases creadas, y comprueba que sigue funcionando.

9.5. Copia las clases anteriores como una nueva versión de las mismas (añade sufijo _v5) y realiza los siguientes cambios:

- Añade un constructor más a la clase Figura2D, a este se le pasa solo un parámetro que se le asigna tanto al alto como al ancho (figura igual alto que ancho).
- Hacer el constructor sin parámetros (ya no se crea por defecto) en este caso tanto alto como ancho igual a cero.
- Añade a la clase Triangulo también un constructor sin parámetros, que invoque al constructor sin parámetros de la clase base (aunque se hace por defecto) y que ponga el estilo a null.
- Añade otro constructor a la clase Triangulo, con un solo parámetro, para aquellos que tienen igual alto que ancho (en este caso el estilo será 'igualBaseAltura').
- Haz un programa similar a los de ejercicios anteriores pero que use las nuevas características incorporadas en este ejercicio.

9.6. Copia las clases anteriores como una nueva versión de las mismas (añade sufijo _v6) y realiza los siguientes cambios:

- Crea una nueva clase TrianColor_v6, hija de la clase Triángulo_v6 que incluye un nuevo atributo privado de tipo String llamado color.
- Esta nueva clase tiene un constructor de 4 parámetros (alto, ancho, estilo, color), además del getter y setter de color.
- Haz un nuevo programa que cree un triángulo de color y llame a métodos definidos en sus clases base.

9.7. Copia las clases anteriores como una nueva versión de las mismas (añade sufijo _v7) y realiza los siguientes cambios:

- Figura2D_v7 tendrá un nuevo constructor, que recibe como parámetro otro objeto de tipo Figura2D_v7 y que copiará todos sus datos.
- Hacer otro constructor análogo para la clase Triangulo_v7.
- Haz un programa que cree un Triangulo_v7 con el constructor de 3 parámetros

creado previamente, y otro triángulo utilizando el nuevo constructor. Mostrar el área de ambos.

9.8. Copia la última versión de las clases Figura2D, Triangulo, Triancolor como una nueva versión de las mismas (añade sufijo _v8) y realiza los siguientes cambios:

- Añadir a la clase Figura2D_v8 un atributo privado llamado nombre de tipo String.
- Añadir el getter/setter de ese campo y modificar los constructores de dicha clase para incorporar como parámetro el nombre de la figura (en el constructor por defecto se le asignará valor null).
- Modificar las clase hija (Triangulo_v8) y nieta (Triancolor_v8) para incluir el nombre en constructores.
- Crear una nueva clase hija de Figura2D_v8 llamada Rectangulo_v8 con constructor con tres parámetros (alto, ancho, nombre), constructor con un dos parámetros (alto igual a ancho y nombre) y un método que devuelve boolean llamado esCuadrado().
- Hacer un programa que cree un ArrayList con 4 rectángulos, algunos cuadrados y otros no, y cuente cuantos hay cuadrados. En el mismo main() crear un triancolor a los que le asignes también el nuevo atributo: nombre y muestres su área.

9.9. A partir de la clase CuentaCorriente que te proporcionará el profesor realiza las siguientes operaciones:

- Estudia los métodos: ‘ingresar’ y ‘retirar’ y añádeles un comentario a cada método explicando su funcionamiento (bonificaciones y comisiones).
- Crea una clase hija llamada CuentaPlazo igual que la anterior, pero sin comisiones. Además, tiene un nuevo atributo, que es la fecha de creación.
- Esta nueva clase ya no permitirá que haya subclases de ella.
- Crea constructor para la nueva clase que incorpore la inicialización de la fecha de creación al día en curso.
- Sobrescribe el método retirar () para que no que no calcule comisiones. Revisa los modificadores de acceso de atributos y quizás tengas que crear algún getter/setter.
- Haz un programa que cree inicialmente una cuenta a plazo y luego, mediante un menú, permita ingresar, retirar y consultar el saldo.

9.10. A partir de la clase MovilPrepago que te proporcionará el profesor realiza las

siguientes operaciones:

- Crea una subclase llamada MovilPlus igual que la anterior pero que al efectuar llamadas no se le aplica ningún coste por el establecimiento de llamada y tiene una nueva funcionalidad llamada videollamada a la que se le pasa los segundos de la videollamada y reduce el saldo de forma similar a 'navegar'. En este caso, cada segundo de llamada son 2 MB de datos (recuerda que los atributos de MovilPrepago son privados, esto puede ser un problema a resolver).
- Crea constructor para la nueva clase, que llame al constructor de la clase padre.
- Sobrescribe el método toString () en la superclase para que muestre el número y el saldo del móvil.
- Añade una nueva subclase de MovilPrepago llamada MovilTarifaPlana¸ en la que se redefine el método navegar para no reducir el saldo y además en el constructor fija el coste de navegación a cero.
- Haz un programa que cree una instancia de MovilPrepago, otra de MovilPlus y otra de MovilTarifaPlana, y realicen diversas operaciones sobre los mismos: llamar, navegar, videollamar, etc. mostrando como se reduce su saldo con el nuevo método toString ().

9.11. Crear una clase llamada Trabajador con los atributos privados: id, nombre, fecha de nacimiento y salario base.

- Dispondrá también de un constructor que inicialice todos sus campos, getters, setters, método toString () y equals (), sabiendo que dos trabajadores son iguales si tienen el mismo 'id'.
- Crear una subclase de Trabajador llamada Asalariado que añade un nuevo atributo llamado salarioFinal y otro llamado horasExtra.
- El constructor de esta nueva clase Asalariado incorpora la inicialización a cero de las horas extra y el salario final igual al salario base.
- La clase Asalariado también incorpora setter y getter para las horas extra y un método llamado calcularSalarioFinal() al que se le pasa a cuanto se paga la hora extra en ese momento y calcula el salario final del empleado siendo su salario base más el importe de las horas extras trabajadas.
- Crear una subclase de Trabajador llamada ConsultorExterno que añade un nuevo atributo llamado horasTrabajadas y salarioFinal.
- El constructor de esta nueva clase ConsultorExterno incorpora la inicialización a cero de las horas trabajadas, salario base y salario final.
- La clase ConsultorExterno también incorpora setter y getter para las horas trabajadas y un método llamado calcularSalarioFinal() al que se le pasa a cuanto se paga la hora a los consultores en ese momento y calcula el salario final del

consultor solo en función de las horas trabajadas (el salario base de estos trabajadores es cero).

- Haz un programa que cree un ArrayList de Asalariados y otro de ConsultoresExternos e introduce “a mano” dos o tres empleados en ambos ArrayList.
  - Después modifica el contenido de ambos ArrayList añadiendo también “a mano” las horas extra/horas trabajadas respectivamente para todos los empleados.
  - Fijar el importe de hora extra a 20 euros y la hora de consultor a 100 euros y modificar de nuevo los ArrayList calculando el salario final de cada trabajador.
  - Finalmente, recorriendo con un for-each ambos ArrayList, obtener el total que gastará la empresa en salarios.
  - ¿Sería útil tener un ArrayList que pudiese contener instancias de ambas clases?

9.12. Realiza un programa con una variable de tipo Figura2D_v8, y sobre ella llama a uno de los constructores de Triangulo_v8. Muestra sus dimensiones y el cálculo del área.

9.13. Copia la última versión de las clases de los primeros ejercicios: Figura2D, Triangulo, TrianColor y Rectangulo (añade sufijo _v9) y realiza los siguientes cambios:

- Crea un método abstracto llamado area () en Figura2D que ha de implementarse en las clases hijas.
- Crea un método precio (float precioMetroCuadrado) en la clase Figura2D, que use el método abstracto anterior.
- Verificar que las clases hijas implementan el método area (). ¿Qué ocurriría si no lo tuviesen implementado?
- Haz un programa que almacene figuras de los tres tipos en un ArrayList y finalmente se recorra el ArrayList con un for-each mostrando el precio de cada figura, suponiendo un precio de 10 euros el metro cuadrado.

9.14. Diseña una clase abstracta llamada Figura3D_v1 con método abstracto volumen (). Crea subclases: Esfera_v1 y PrismaRectangular_v1 que implementen el método de la superclase. Incorpora los atributos que creas necesarios a las tres clases. Haz un programa que cree una instancia de cada una de las 2 figuras y muestre cuál de ellas ocupa más.

9.15. Haz una nueva versión de las clases del ejercicio anterior (añádele el sufijo _v2). Añádele a la clase base el método abstracto superficie (). Crea una nueva clase Cilindro y haz que implemente los métodos de la superclase. Haz un programa que permita crear una instancia de cada una de las 3 figuras y nos muestre cuál tiene más superficie.

9.16. Convierte la clase abstracta Figura3D_v1 en una interfaz Figura3D_v3 con las mismas características. Rehaz las clases Esfera y PrismaRectangular (tendrán sufijo v3) para adaptarlas a la nueva interfaz. Haz un programa que cree una instancia de cada una de las 2 figuras y muestre cuál de ellas ocupa más.

9.17. Haz una nueva versión de la interfaz y clases del ejercicio anterior (añádele el sufijo _v4). Añádele a la interfaz el método superficie (). Crea una nueva clase Cilindro_v4 y haz que implemente los métodos de la interfaz. Haz un programa que permita crear una instancia de cada una de las 3 figuras y nos muestre cuál tiene más superficie.

- Cuestión 1: ¿Al añadir el nuevo método a la interfaz, si no modificamos las clases que la implementan ¿se produce algún error? ¿Por qué? ¿Cómo evitamos esta situación?
- Cuestión 2: Supón que añades a la interfaz un nuevo método llamado ocupaMasque va a ser igual para todas las clases que implementen la interfaz ya que es una comparación del volumen, tenga la forma que tenga ¿qué opciones tenemos para no tener que implementarlo en todas las clases?

9.18. Crea una clase llamada Consola con un método estático sobrecargado llamado leerEntero() que solicite al usuario que teclee un valor entero, cumpliendo las siguientes características:

- Si no se le pasa ningún parámetro, no tiene requisitos, es simplemente un nextInt().
- Si se le pasa un parámetro de tipo texto, escribe ese texto antes de solicitar el valor. Ejemplo: leerEntero (“Introduzca su edad”);
- Si se le pasa un parámetro tipo texto y dos enteros, garantizará que el valor tecleado esté comprendido entre ambos. Ejemplo: leerEntero (“Introduzca su edad”, 0, 120);
- Si se le pasan dos enteros, garantizará que el valor tecleado esté comprendido entre ambos, pero no muestra texto de instrucciones previo. Ejemplo: System.out.println (“Introduzca su edad”); leerEntero (0, 120);

Finalmente, haz un programa que pruebe todas las variantes del método.

9.19. Haz un programa con un menú que permita gestionar la cola de espera de un médico. Hay tres tipos de pacientes: los que vienen a consulta (se le pide al usuario nombre, edad, motivo de la consulta), los que viene por recetas (se le pide: nombre, edad, lista de medicamentos) y el que viene a revisión (se le pide nombre, edad y fecha de la visita anterior).

- Las tarifas del médico son: Consulta: 100 eur. Recetas: 5 eur por cada unidad. Revisión: 30 eur para mayores de 65 años, 50 eur para resto.
- Crear una clase para cada tipo de paciente en el propio archivo del programa con los constructores necesarios y el método de facturar() en cada una de las clases. Implementa herencia si lo crees necesario.
- El programa tendrá un menú para:
  a) Registrar la llegada del paciente: se le preguntará por qué viene al médico y se le piden sus datos.
  b) Llamar a consulta (por orden de llegada). Se le cobra la tarifa correspondiente.
  c) Consultar total facturado hasta ese momento.

Puedes crear una clase Clínica, que tendrá un ArrayList de Pacientes, o bien definir ese ArrayList en el programa como variable global y no tener la clase Clínica.

9.20. Haz un programa con un menú que permita gestionar un parking.

- El parking tiene 100 plazas y pueden aparcar cualquier tipo de 3 tipos de vehículos distintos: Vehículos en general, Furgonetas y Autobuses.
- Todos los vehículos pagan 4 céntimos por minuto, pero las furgonetas pagan además un suplemento de 20 céntimos por cada metro de su longitud y los autobuses pagan un suplemento de 25 céntimos por asiento.
- El menú del programa deberá permitir:
  a) Entrada de un vehículo. Se le pide al usuario la matrícula, tipo de vehículo y datos adicionales para el cálculo de la estancia (longitud, número de asientos...).
  b) Salida del vehículo. Se le pide al usuario la matrícula, se calcula el importe a pagar y libera la plaza.
  c) Mostrar la lista de vehículos en el parking con la matrícula, tipo de vehículo y fecha/hora de llegada (piensa en el método toString). Al final número total de plazas ocupadas.
  d) Salir
- Puedes hacer coste 4 céntimos por segundo (en vez de por minuto) para

probarlo.

- No hay el concepto de número de plaza, los coches van aparcando donde quieren.
- Mantén la mayor cantidad de información (datos y cálculos) en las clases, no en el programa. Puedes crear las clases en el mismo archivo que el programa (por comodidad) con el modificador de acceso por defecto.
- El parking será un ArrayList. Para localizar un vehículo (en la opción de menú de Salida del vehículo) emplea ArrayList.indexOf y ello te puede implicar definir equals en alguna clase.

Al igual que en el ejercicio anterior, puedes crear una clase Parking, que tendrá un ArrayList de Coches, o bien definir ese ArrayList en el programa como variable global y no tener la clase Parking.

9.21. Crea una clase Bicicleta de la que deseamos mantener los siguientes datos: marca, modelo, precio y descuento. Se pide crear el constructor, getters y setters, método toString (), equals (), un método que devuelva el precio con el descuento aplicado y finalmente un método que fije el descuento a aplicar. Este último método estará sobrecargado de la siguiente forma:

- fjarDescuento () ☾ (se le hace 10% y dura ese descuento 1 mes)
- fjarDescuento (double d) ☾ (se le hace d %, 1 mes)
- fjarDescuento (double d , int n) ☾ (se le hace d %, n meses)

Haz un programa sencillo que defina una o dos instancias de bicicletas y use los métodos creados Notas:

- Dos bicicletas son iguales si tienen la misma marca y modelo.
- Si se fija un descuento, se elimina el descuento que pudiera haber anteriormente.
- Piensa si es necesario incorporar algún atributo adicional, para que, cuando ejecutemos el método de ver el precio final (con descuento aplicado), sepa si tiene que aplicar algún descuento o no.

9.22. A partir de las clases siguientes:

```
public abstract class PoligonoRegular {
	double tamañoLado;
	abstract double area();
	abstract int getCantidadLados();
	double perimetro() {
		return getCantidadLados() * this.tamañoLado;
		PoligonoRegular(double tl) {
			this.tamañoLado = tl;
		}
```

```
        }
    }
    class Pentagono extends PoligonoRegular {
        Pentagono(double t) {
            super(t);
        }
        @Override
        int getCantidadLados() {
            return 5;
        }
        @Override
        double area() {
            return 1.72048 d * Math.pow(this.tamañoLado, 2);
        }
    }
    class Hexagono extends PoligonoRegular {
        Hexagono(double t) {
            super(t);
        }
        @Override
        int getCantidadLados() {
            return 6;
        }
        @Override
        double area() {
            double lado = this.tamañoLado;
            double apotema = Math.sqrt((lado * lado) - ((lado / 2) * (lado / 2)));
            return lado * apotema * 3;
        }
    }
```

a) Prueba a crear instancias de hexágonos y pentágonos sobre variables de tipo PoligonoRegular y comprueba mediante alguna calculadora online que calcula correctamente su área y perímetro.

b) ¿Por qué PoligonoRegular es abstracta?

c) ¿El método área() podría ser no abstracto?

d) ¿Es posible crear una clase hija de PoligonoRegular sin desarrollar el método área() ?

e) ¿Puedo crear instancias de PoligonoRegular ?

f) ¿Por qué el método perímetro() no es abstracto si no se calcula igual para cada tipo de polígono regular? (pentágono es lado por 5, hexágono es lado por 6, etc.)

g) ¿El siguiente código es correcto? Explica por qué funciona.

```
ArrayList <PoligonoRegular> listaPoligonos = new ArrayList <>();
for (PoligonoRegular p : listaPoligonos)
System.out.println(p.area());
```

h) Si no existiesen las clases abstractas y el método área() lo definiésemos en las

clases hijas, ¿Funcionaría el código anterior? ¿Por qué?

9.23. Copia el ejercicio del capítulo anterior de la lista de espera del médico y redefine la superclase Paciente como abstracta haciendo el método facturar() abstracto, ya que se implementa en las tres clases hijas.

9.24. Pensando que en el futuro implementemos el juego de ajedrez para dos jugadores, se desea crear una clase abstracta llama PiezaAjedrez, con dos atributos enteros llamados fila y columna que representan sus coordenadas en el tablero (valores entre 0 y 7) y un método abstracto llamado mover () al que se le pasan como parámetro la fila y columna destino de un movimiento. El método devuelve true si el movimiento se puede realizar o false si es un movimiento erróneo. Implementa esa clase y sus subclases AlfilAjedrez y TorreAjedrez. Para simplificarlo, vamos a pensar en movimientos en un tablero vacío, es decir solo con una pieza, la que se está movimiento.

- Haz un programa que permita al usuario mover una sola pieza (al comenzar elegirá Alfil o Torre) por el tablero, partiendo de la posición 0,0, indicando las coordenadas destino del movimiento que quiere hacer cada vez, terminando el programa cuando introduzca fila -1.
- El programa tendrá una función que presente por pantalla la situación del tablero.
- Por comodidad, puedes hacer las clases dentro del mismo fichero que el programa.

9.25. Modifica la clase PiezaAjedrez (versión _v2) para incluir métodos ¿estáticos? para que el usuario introduzca la columna como letra (a-h) y la fila (entre 1 y 8) y los convierta a los valores usados previamente (entre 0 y 7). Esto obligará a generar una nueva versión del programa anterior, para que el usuario introduzca a-h y 1-8 como valores destino del movimiento.

9.26. Desarrolla una interfaz llamada Ciclista con un método llamado recorrer() al que se le pasa un número de kilómetros y una cadena con tipo de terreno y devuelve los segundos que tarda en recorrerlo. Una interfaz llamada Nadador con un método llamado nadar a la que se le pasan metros y devuelve los segundos en recorrerlo, y por último otra interfaz llamada Saltador con un método saltarAltura que no recibe parámetros y que devuelve los centímetros saltados.

- Desarrolla una clase Delfin que implemente la interfaz Nadador. El tiempo en recorrer una distancia es aleatorio entre 40km/h y 70km/hora
- Desarrolla una clase BallenaAzul que implemente la interfaz Nadador. El tiempo

en recorrer una distancia es de 10km/h para las mayores de 5 años y 13km/h para las menores.

- Desarrolla una clase TriAtleta que implemente las tres interfaces, con los criterios que tu consideres (pueden devolver valores aleatorios entre unos mínimos y máximos que tú decidas o tener en cuenta otros parámetros como edad, sexo, etc.). Realiza un programa sencillo que cree instancias de delfines, ballenas azules y triatletas y use los métodos desarrollados.

9.27. Empleando las tres clases definidas en el ejercicio anterior (Delfín, BallenaAzul y TriAtleta) y la interfaz Nadador, realiza un programa que contenga un ArrayList de nadadores (polimorfismo de interfaces) con tres elementos, uno de cada tipo, esto es un Delfín, una BallenaAzul y un TriAtleta. A continuación, recorre el ArrayList con un bucle for...each, mostrando cuánto tarda cada uno de ellos en recorrer un kilómetro.

9.28. Duplica las interfaces y las clases del ejercicio anterior (añadiéndoles el sufijo _v2) Modifica la interfaz Saltador_v2 añadiéndole el método saltaPertiga que recibe como parámetro una altura en centímetros y devuelve true si ha logrado el salto y false si no lo ha logrado ¿Qué ocurre con la clase TriAtleta_v2? Desarrolla saltaPertiga como método default en la interfaz de forma que por defecto devuelva false.

- Haz una nueva versión de TriAtleta (sufijo _v2b) que implemente saltaPertiga con este criterio: para saltos de más de 6 metros devuelve false, entre 5 y 6 metros devuelve true la mitad de las veces y por debajo de 5 metros devuelve siempre true.
- Haz un programa que cree una instancia de Triatleta_v2 y otra de Triatleta_v2b y que muestre el resultado de ambos atletas saltando: 100cm, 550cm, 560cm, 580cm, 700cm.

9.29. Añade el sufijo _v2 a las clases de capítulos previos: movilPrepago, movilTarifaPlana, movilPlus y crea una interfaz llamada PrePagable que estas clases deberían implementar. Incluye en la interfaz todos los métodos que creas oportuno y, si es necesario, alguno puede ser método por defecto.

9.30. Se desea hacer la gestión de las habitaciones de un hotel. Todas las habitaciones tienen un número de habitación y un proceso de check-in y check-out. En el hotel hay estas habitaciones:

1. 3 habitaciones Lowcost (cuesta 50 euros/dia todo el año).
2. 10 habitaciones dobles. Tienen una tarifa normal de 100 euros/dia y un incremento del 20% si el día de salida es abril, julio o agosto.

3. 5 habitaciones Suite. 200 euros/día con 20% de descuento para estancias de 10 o más días.

- El programa inicialmente meterá las 18 habitaciones en un ArrayList y las marcará como “no ocupadas”.
- El programa tendrá un menú para hacer check-in entre las habitaciones libres, check-out entre las ocupadas y listar habitaciones libres y ocupadas.
- El check-in es común para todas las habitaciones, consiste en marcar la habitación como ocupada.
- El check-out consiste en marcar la habitación como libre y calcular el importe a pagar en función de los días de estancia (quizás sea necesario almacenar la fecha de llegada en el momento del check-in)
- Mantener toda la información en las clases más que en el programa que las utiliza.
- Sugerencia: Para probar el programa, al hacer el check-out, puedes considerar cada día como un segundo, para que los datos sean más reales, así han pasado 3 segundos, considerar 3 días.
- La superclase Habitacion debe tener: public abstract double checkOut ();

Cuestión 1: ¿Es útil que el método checkOut sea abstracto o los tres tipos de habitación podrían compartir un código común?

Cuestión 2: ¿Es útil que el método toString() en la clase Habitacion?

Puedes crear una clase Hotel, que tendrá un ArrayList de Habitaciones, o bien definir ese ArrayList en el programa como variable global y no tener la clase Hotel.